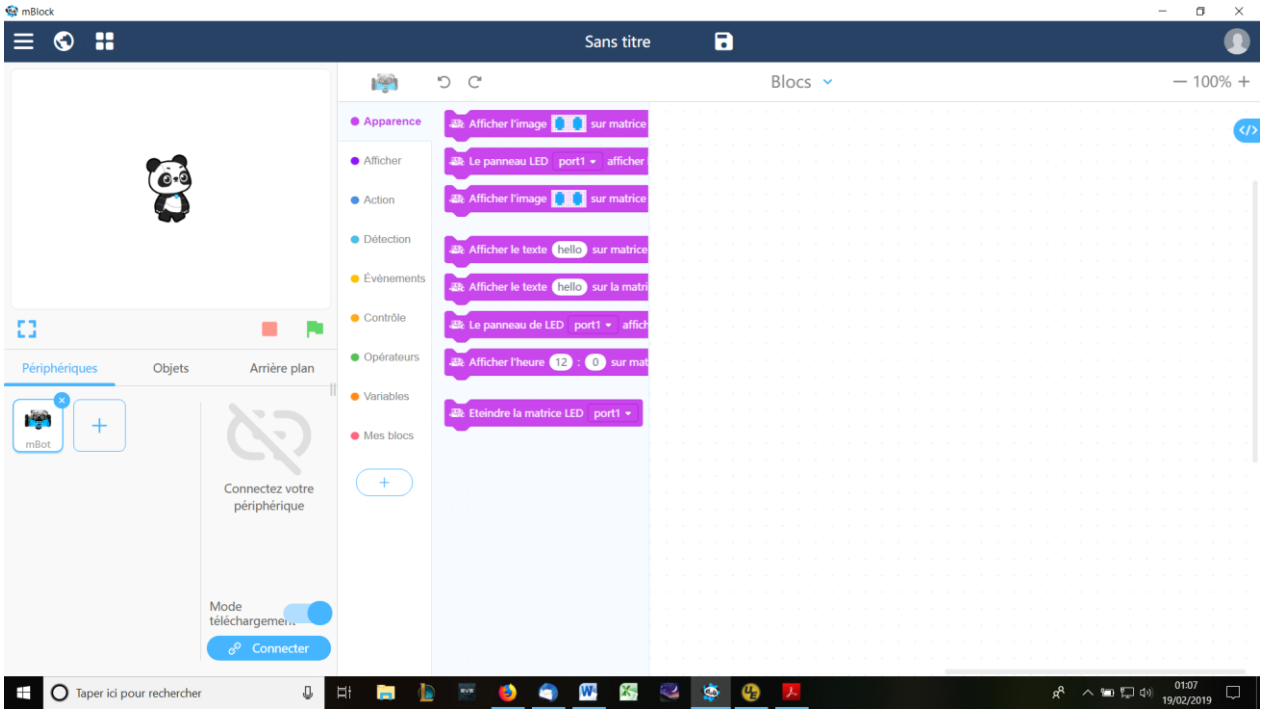
	**Initiation à la robotique**	**SNT**
	Problèmes de robotique avec le robot mBot	TP

Nous allons utiliser le robot mBot, d'abord en programmation par blocs en utilisant le logiciel mBlock, puis en modifiant le code Arduino directement.

1 Généralités sur le robot mBot

Sous mBlock, supprimer le périphérique « Codey » et ajouter « mBot ».
Nous allons travailler de façon à déverser nos programmes dans la mémoire du robot, c'est-à-dire sur mBlock avec l'option « Mode téléchargement »

L'écran devrait ressembler à :



2 Exercice 1 : Découverte du robot

Ce premier exercice permet de vérifier le bon fonctionnement des moteurs du robot ; Nous en aurons besoin.

Consigne : Faire avancer le robot en ligne droite pendant 5 secondes

Une fois le programme créé, le déverser dans le robot pour le tester.

3 Exercice 2 : Différents types de rotation

Consigne 1 : Le robot doit tourner autour d'un point, vers la droite ou vers la gauche. Aucune contrainte angulaire n'est imposée.

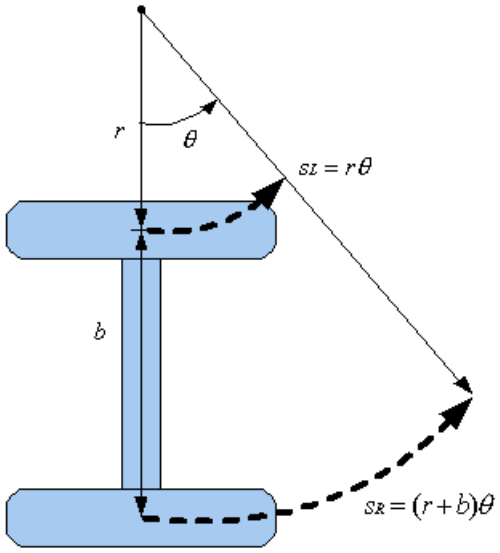
Consigne 2 : Le robot doit tourner autour de l'une de ses roues.

Consigne 3 : Le robot doit tourner autour du point milieu entre ses roues.

Consigne 4 : La trajectoire du robot doit être un carré ou un rectangle. Il doit revenir à peu près à sa position de départ.

Consigne 5 : Le robot doit tourner avec un rayon de braquage égal à la distance entre les roues.

Consigne 6 : Le robot doit tourner avec un rayon de braquage égal 2 fois la distance entre les roues.



4 Exercice 3 : les commandes Arduino

Voici un programme simple en Arduino C pour le robot mBot :

```
// generated by mBlock5 for mBot
// codes make you happy

#include <MeMCore.h>
#include <Arduino.h>
#include <Wire.h>
#include <SoftwareSerial.h>
```

```
double currentTime = 0;
double lastTime = 0;
double getLastTime(){
  return currentTime = millis() / 1000.0 - lastTime;
}
MeDCMotor motor_9(9);
MeDCMotor motor_10(10);

void _delay(float seconds) {
  long endTime = millis() + seconds * 1000;
  while(millis() < endTime) _loop();
}

void setup() {
  lastTime = millis() / 1000.0;
  while(!(getLastTime() > 5))
  {
    _loop();
    motor_9.run(-255);
    motor_10.run(255);
  }
  motor_9.run(0);
  motor_10.run(0);
}

void _loop() {
}

void loop() {
  _loop();
}
```

Consigne 1 : Etudier le programme et expliquer comment va réagir le robot. Le tester ensuite.

Consigne 2 : Modifier le programme pour obtenir la rotation 2. Le tester ensuite.

Consigne 3 : Modifier le programme pour obtenir la rotation 4. Le tester ensuite.

5 Exercice 4 : Suivi de mur (ou le gigue du robot qui a trop bu)

Notre objectif est de faire en sorte que le robot suive un mur (par exemple celui à sa droite) à une distance de 10 cm, avec une marge d'erreur inférieur à 5 mm pendant 10 secondes.
Pour cela on va utiliser le capteur à ultrasons.

Consigne 1 : Modifier le robot pour que son capteur à ultrasons lui permette de suivre un mur sur sa droite.

Consigne 2 : Réaliser le programme en blocs. Le tester ensuite. Critiquer le mouvement du robot.

Consigne 3 : Etudier le programme suivant et expliquer comment va réagir le robot. Le tester ensuite. Critiquer la trajectoire obtenue.

```
// generated by mBlock5 for mBot
// codes make you happy
#include <MeMCore.h>
#include <Arduino.h>
#include <Wire.h>
#include <SoftwareSerial.h>
```

```
double currentTime = 0;
double lastTime = 0;
double getLastTime(){
  return currentTime = millis() / 1000.0 - lastTime;
}
MeUltrasonicSensor ultrasonic_3(3);
MeDCMotor motor_9(9);
MeDCMotor motor_10(10);


void _delay(float seconds) {
  long endTime = millis() + seconds * 1000;
  while(millis() < endTime) _loop();
}

void setup() {
  lastTime = millis() / 1000.0;
  while(!(getLastTime() > 10))
  {
    _loop();
    if(ultrasonic_3.distanceCm() < 9.5){
        motor_9.run(-1 * (75 / 100.0 * 255));
        motor_10.run(90 / 100.0 * 255);

    }else{
        if(ultrasonic_3.distanceCm() > 10.5){
            motor_9.run(-1 * (90 / 100.0 * 255));
            motor_10.run(75 / 100.0 * 255);

        }else{
            motor_9.run(-1 * (90 / 100.0 * 255));
            motor_10.run(90 / 100.0 * 255);

        }

    }

  }
  motor_9.run(0);
  motor_10.run(0);

}

void _loop() {
}

void loop() {
  _loop();
}
```

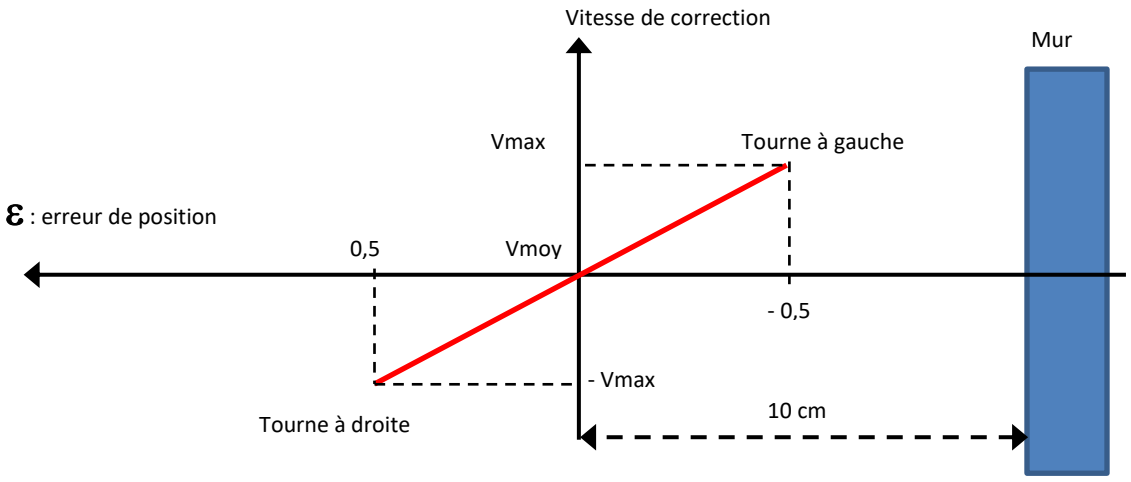
6 Exercice 5 : Tentative d'amélioration du comportement.

L'observation du comportement montre que :

- lorsqu'on augmente la vitesse de correction, on obtient une grande instabilité (fréquence de la gigue) mais une faible amplitude de l'erreur de position par rapport au mur. Le robot avance lentement.
- lorsque l'on diminue la vitesse de correction, on obtient une meilleure stabilité mais une plus grande amplitude de l'erreur de position par rapport au mur. Le robot avance plus vite.

La vitesse de correction a donc une influence sur la position par rapport au mur et sur la vitesse globale de déplacement du robot. Dans l'algorithme précédent, on a corrigé la trajectoire en imposant une vitesse de correction identique quelle que soit la position du robot par rapport au mur.

Il serait bon de pouvoir moduler cette vitesse de correction en fonction de la position du robot.
Une solution envisageable est de calculer une loi de vitesse permettant d'augmenter cette vitesse de correction lorsque le robot s'éloigne de la position voulue et de la diminuer lorsque le robot se rapproche de la position voulue.



Ce type de correcteur s'appelle « Correcteur proportionnel ».

L'expression de la vitesse de correction est donc : $Vc = \frac{Vmax}{0,5} \times \frac{\varepsilon}{0,5} = 4.\varepsilon.Vmax$

On appliquera donc les vitesses suivantes sur les moteurs :

- Moteur droit : Vmoy - Vc
- Moteur gauche : Vmoy + Vc

Remarque : Le coefficient de correction est généralement noté Kp (coefficient du correcteur proportionnel). Ici, Kp=4Vmax

Consigne 1 : mettre en œuvre une correction proportionnelle du mouvement du robot.

Consigne 2 : Modifier le programme Arduino C de l'exercice 4 de manière à ce que le setup ressemble à celui-ci-dessous. Le tester. Critiquer la trajectoire du robot.

```
void setup() {
  lastTime = millis() / 1000.0;
  moyenne = 10;
  Kp = 40;
  Vmoy = 80;
  while(!(getLastTime() > 10))
  {
    _loop();
    delta = ((ultrasonic_3.distanceCm() - moyenne));
    Vc = Kp * delta;
    motor_9.run(-1 * ((Vmoy + Vc) / 100.0 * 255));
    motor_10.run((Vmoy - Vc) / 100.0 * 255);
  }
  motor_9.run(0);
  motor_10.run(0);
}
```

6 Exercice 6 : Améliorations plus complexes du comportement.

Lorsque l'erreur de position est faible, la correction est faible. Si l'erreur persiste, il faut longtemps au correcteur proportionnel pour agir. On serait tenté d'augmenter la valeur de Kp mais dans ce cas, le robot se met à osciller.
L'idée dans ce cas est de calculer le cumul des erreurs pendant un certain temps et le multiplier par un coefficient pour ajuster la correction de vitesse :

$$Vc = Kp.\varepsilon + Ki.\Sigma(\varepsilon.\Delta t)$$

Ce type de correcteur s'appelle « Correcteur intégral ».
La somme des erreurs courantes est renouvelée à chaque itération du programme (Δt). Il s'agit donc d'un calcul intégral. Le temps entre 2 itérations étant constant, Δt peut faire partie du coefficient intégral Ki (d'où une valeur très petite de Ki en général, 0.001 par exemple). Ce dernier pourra être ajusté par tâtonnements.

Le facteur intégral est l'histoire cumulative des erreurs et donne au correcteur un moyen de les corriger quand elles persistent pendant une longue période de temps.

Consigne 1 : mettre en œuvre une correction proportionnelle et intégrale du mouvement du robot.

Consigne 2 : Modifier le programme Arduino C de l'exercice 5 de manière à ce que le setup ressemble à celui-ci-dessous. Le tester. Critiquer la trajectoire du robot.

```
void setup() {
  lastTime = millis() / 1000.0;
  moyenne = 10;
  Kp = 40;
  Ki = 0.001;
  Vmax = 80;
  while(!(getLastTime() > 20))
  {
    _loop();
    delta = ((ultrasonic_3.distanceCm() - moyenne));
    somme += delta;
    Vc = Kp * delta + Ki * somme;
    motor_9.run(-1 * ((Vmax + Vc) / 100.0 * 255));
    motor_10.run((Vmax - Vc) / 100.0 * 255);
  }
  motor_9.run(0);
  motor_10.run(0);
}
```

On peut encore faire mieux mais c'est encore plus complexe !!